0. Attacker Service

The objective of this project is to develop scripts that analyze logs from web application attacks. These logs contain crucial information that can help us understand the nature of the attacks, identify the attackers, and uncover the vulnerabilities exploited. By scrutinizing these logs, we can gather actionable intelligence to strengthen our web application's security posture.

Which service did the attackers use to gain access to the system? Write a script that scan the logs and help you figure what service was used

```
┌──(imenⓀhbtn-lab)-
[…/web_application_security/0x0c_web_application_foresics]
└─$ ./0-service.sh
34806 pam_unix(sshd:auth):
  20339 Failed
  14478 Invalid
    214 Address
    200 pam_unix(sshd:session):
    169 reverse
    118 Accepted
     44 Did
     20 error:
     20 Server
     10 subsystem
      9 syslogin_perform_logout:
      7 Received
      5 PAM
      5 Jax
      2 Bad
      1 new
      1 changed
      1 change
      1 Kayn
      1 Exiting
```

Command Breakdown:

```
awk '{print $6}' auth.log | sort | uniq -c | sort -nr
```

1. **`awk '{print $6}' auth.log`**:
    - This extracts the **6th column** from the `auth.log` file. In most log formats, the 6th column represents the **username** involved in an authentication attempt (though this can vary depending

on the system and log format). For example, in a typical SSH login log entry, it could be the username that attempted to log in.

2. **`sort`**:
   - This sorts the output of the previous `awk` command in **alphabetical order**. Sorting is necessary before using `uniq -c` to count occurrences.
3. **`uniq -c`**:
   - This command removes duplicates from the sorted list and prepends the count of each unique value (username) from the previous step. This shows how many times each username appears in the log.
4. **`sort -nr`**:
   - Finally, this sorts the output by the **numerical count** in **reverse order** (`-nr`). This means the usernames with the highest number of authentication attempts will appear at the top.

What the command does:

The entire pipeline extracts usernames from the `auth.log` file, counts the number of occurrences for each unique username (indicating the number of authentication attempts), and then sorts the usernames by the number of attempts in descending order.

Example output:

If you ran this command on a sample `auth.log`, the output might look like this:

```
 50 root
 35 user1
 10 user2
  5 user3
```

This would indicate that the user `root` had 50 authentication attempts, `user1` had 35, `user2` had 10, and so on.

Use case:

This is useful for identifying suspicious activity, such as brute force attempts, where multiple failed login attempts are made by a specific user. It helps quickly spot which usernames are being targeted most frequently.